Express Mail Label
No. EL 849003485 US

Attorney Docket No.
14406US03

APPENDIX E

Title : Radio Frequency Local Area Network

Inventors: Meier, et al.

Attorney Docket No. 14406US03

BRIDGING LAYER SPECIFICATION

Introduction. 2
Functional Requirements. 2
Bridge Packet Definitions. 2
Constants, timers and data structures. 5
State Transition Logic - notation and assumptions. 7
Root Resolution Protocol. 8
Attaching to the Spanning Tree. 11
Reliable Flooding Mechanism. 16
Spanning Tree Link Maintenance and Recovery. 16
Network Routing. 21
Address Resolution and Maintenance. 22
HELLO Timing. 25
Sleeping terminal support. 28

Introduction.

This document, in conjunction with the SST network frame format specification, defines the routing layer protocol used with the SST network.

Functional Requirements.

The bridging layer:

- selects a root bridge node.

- organize nodes in the network into an optimal spanning tree rooted at the root bridge.

- routes terminal-to-terminal data traffic along branches of the spanning tree. Note that a host computer is treated as a terminal. A routing table is maintained in each bridge node for all upstream nodes. The path to downstream nodes is inherent in the structure of the spanning tree.

- provides a service for storing packets for SLEEPING terminals.

- propagates lost node information throughout the spanning tree.

- maintains the spanning tree links.

- distributes network interface addresses.

- organizes nodes into logical coverage areas on radio channels.

Bridge Packet Definitions.

Bridge Control Byte Fields.

Bridge header format - This field is used to define the format of the bridge header.

00 = multihop. This is the normal bridge header format. The bridge header includes source and destination address fields.

01 = point-to-point. The bridge header does NOT include source and destination address fields.

Packet type - This field is used to specify the bridging layer packet type.

Bridge parms - If this bit field is set ON then optional bridging layer parameters immediately follow the bridge header.

RSPRQ - This field can be used to request an end-to-end bridging layer response packet. Normally this field should be set ON for ATTACH, RARP and ARP request packets, and should be set OFF for all other bridging layer packets.

ATTI - A bridge node will set this bit ON in an ATTACH.request packet whenever the source node is not in the bridge's routing table. The bit value in an ATTACH.response packet follows the state of the bit in the associated ATTACH.request packet received by the root node. If a terminal receives an ATTACH.response packet with the ATTI bit set ON, it is a positive indication that the terminal was detached and has reattached to the network.

Protocol - This field is used to indicate the presence and type of higher layer data.

000 = no higher layer data is contained in the packet.
001 = an LLC message is contained in the packet.
010 = an NNMP message is contained in the packet.

Bridge Packet Types.

DATA.request.

DATA.request packets are general purpose bridging layer packets used to send higher layer data and/or bridging layer parameters.

A child node can periodically send a (possibly empty) DATA.request packet to its parent to ensure that it is still attached to the spanning tree. Note that an ATTACH.request packet, with the ATTI bit set ON, is generated by a bridge node whenever the bridge node receives a downstream DATA.request packet and the source of the DATA packet is not in the bridge's routing table.

HELLO.request.

A HELLO.request packet is used to solicit unscheduled HELLO.response packets from bridge nodes. A HELLO.request packet can be broadcast by an unattached node when it wants to quickly reattach to the network.

HELLO.response.

HELLO.response packets are sent periodically at calculated time intervals by all bridge nodes. In addition, bridge nodes will broadcast a HELLO.response packet whenever a HELLO.request packet is received. HELLO.response packets are used to build the spanning tree and are used to advertise pending message information and lost node information.

ATTACH.request.

A node transmits an ATTACH.request packet, with the RSPRQ bit set ON, to attach to the network. In addition, a node must transmit an ATTACH.request packet at least once per ATTACH_TIMEOUT time period to maintain its path in the network. All ATTACH.request packets are implicitly forwarded to the root bridge node. If the RSPRQ bit is set ON the root bridge node will return an ATTACH.response packet.

Higher layer data can be piggybacked onto ATTACH.request packets by setting the bridging layer destination address to the 16-bit address of the node for which the data is intended. If data is piggybacked onto an ATTACH.request packet, the bridging layer will split the ATTACH packet into separate ATTACH and DATA request packets as soon as the next hop to the destination address is not on the path to the root node (i.e. the first upstream hop). The destination address of the generated ATTACH.request packet is the well-known address of the root node.

If a bridge node receives a downstream DATA.request packet and the source node is not in the bridge's routing table, then the bridge node will automatically generate an ATTACH.request packet, with the ATTI bit set ON, for the source node and forward it to the root node. The source address in the generated ATTACH.request packet is the same as the source address in the DATA packet. (Note that the DATA packet can simply be converted to an ATTACH packet.)

ATTACH.response.

The root node will return an ATTACH.response packet to the source node in the associated ATTACH.request packet if the RSPRQ bit in the request packet is set ON. The ATTI bit will be set ON if the originating node was not fully connected to the network (i.e. if the node was not in the routing table of a bridge node on the path of the request packet).

Detach.request.

DETACH.request packets are used to notify the network that a node has detached. DETACH.request packets can be reliably broadcast throughout the spanning tree by using the reliable flooding mechanism described below. If a DETACH.request packets includes a decedent list, then all nodes in the decedent list and the bridging layer source node are purged from the routing table of nodes which receive the request.

Address Resolution Packet (ARP).

An address resolution packet is used to acquire the 16-bit network address of a destination node, when only the alias or 48-bit identifier of the node is known. 16-bit network addresses are cached by the bridging layer. An ARP packet is generated automatically by the bridging layer whenever it receives a send request for which the destination is not in its cache.

Reverse Address Resolution Packet (RARP).

A RARP packet is used to set or change the alias and/or 48-bit long identifier of a device and to acquire a 16-bit network address.

Constants, timers and data structures.

ROOT_ADDRESS = hex 2000;

HELLO_RETRY = 5;
MAX_HELLO_LOST = 8;
HELLO_SLOT_SIZE = .020 seconds.
HELLO_MOD_VAL = 67.
AVG_HELLO_SLOTS = 100.
MIN_HELLO_SLOTS = HELLO_MOD_VAL.
MAX_HELLO_SLOTS = (HELLO_MOD_VAL * 2) - 1.
AVG_HELLO_PERIOD = HELLO_SLOT_SIZE * AVG_HELLO_SLOTS seconds;
MIN_HELLO_PERIOD = HELLO_SLOT_SIZE * MIN_HELLO_SLOTS seconds;
MAX_HELLO_PERIOD = HELLO_SLOT_SIZE * MAX_HELLO_SLOTS seconds;
HELLO_TIMEOUT = HELLO_RETRY * AVG_HELLO_PERIOD + 1 seconds;
S_HELLO_WAIT = .5 seconds;
MAX_HELLO_OFFSET = 200 milliseconds;

BRG_RSP_TIMEOUT = 10 seconds;

HOLD_DOWN_TIME = MAX_HELLO_PERIOD*(MAX_HELLO_LOST+2) seconds;
DEF_SAVE_CNT = 3;
MAX_SAVE_CNT = 5;
AWAKE_TIME_UNIT = .1 seconds.

ADDRESS_TIMEOUT = 30 minutes.
MAX_ADDRESS_LIFE = 120 minutes.

NETWORK_TIMEOUT = 12 minutes.

ROUTE_TIMEOUT = 10 minutes.
ATTACH_TIMEOUT = 8 minutes.

UT_OOR_TIME = 20 seconds; (Unattached terminal out-of-range list entry timeout)
UB_OOR_TIME = 100 seconds; (Unattached bridge out-of-range list entry timeout)
AT_OOR_TIME = 100 seconds; (Attached terminal out-of-range list entry timeout)
AB_OOR_TIME = 300 seconds; (Attached bridge out-of-range list entry timeout)

UT_OOR_AGED_TIME = 300 seconds; (Attached terminal out-of-range list aged entry timeout)
UB_OOR_AGED_TIME = 500 seconds; (Attached bridge out-of-range list aged entry timeout)
AT_OOR_AGED_TIME = 500 seconds; (Attached terminal out-of-range list aged entry timeout)
AB_OOR_AGED_TIME = 1000 seconds; (Attached bridge out-of-range list aged entry timeout)
B_ENABLE_WAIT = 20 seconds;
R_IDLE_TIME = 60 seconds;
MAX_RARP_RETRY = 10;
MAX_ATTACH_RETRY = 5;
B_TIMEOUT; (Bridging layer timeout value - a function of the cost to the root.)

HOP_COST_ETHER = 20;
HOP_COST_485LAN = 40;
HOP_COST_SSRADIO = 125;
HOP_COST_BUS = 0;

OOR_COST = 75;

A **hello-timer** is used to measure a HELLO learning period.

An **inactivity-timer** is used to measure periods of network inactivity.

A **path-timer** is used to measure the maximum round-trip path delay for bridging layer packets which require a response.

In any enabled state, all nodes, except the root, maintain two lists: an **in-range list** and an **out-of-range list.**

An entry in the in-range list contains the network address of a bridge node which was the source of a HELLO packet, the port used to communicate with the bridge node, an aging factor, and path cost information. Each entry in the in-range list is aged so that it is discarded if no HELLO packet is received from the associated bridge within HELLO_TIMEOUT seconds. The aging factor for an entry in the in-range list is reset to zero whenever a HELLO packet is received. The aging factor for the entry of the parent node is reset to zero whenever any packet is received from the parent. In-range entries are classified as OLD after two HELLO periods pass without receiving a HELLO packet from the bridge specified in the entry.

An entry in the out-of-range list contains the network address of a bridge node, the port used to communicate with the bridge, and an aging factor. The out-of-range list is used to remember failed paths when a node is attempting to attach to the network. Entries in the list are aged to indicate how long entries have been in the list. Out-of-range entries are classified as OLD after two HELLO periods in the list. Out-of-range entries are classified as AGED after OOR_TIME seconds in the list. AGED entries which are also in the in-range list are not selected as a potential parent node unless no other in-range entries are available. AGED entries are discarded after OOR_AGED_TIME seconds have passed.

Each node maintains a **current_root** variable, which is initialized to NULL. The variable contains the priority, root sequence, and ROOT ID the currently active root node, where ROOT ID is a 48-bit device ID and/or alias. The variable is used to remember the current root

node. Attached nodes detach and go into an initial unattached state, without a network address, when the root node changes.

In general, an entry is added/updated in the in-range list of an enabled, attached or unattached node whenever a HELLO packet, with the priority, sequence, and identifier of the current root node, is received, and the source bridge is not in the node's out-of-range list.

Each node, except the root, maintains a **current_parent** variable, which contains the network address and path cost information of the current parent of the node when the node is in an attached state.

MSG_PENDING is (conceptually) a global state variable which is true when an incoming message is expected.

State Transition Logic - notation and assumptions.

Assumptions.

The state transition logic in this document and the SST network assumes the following:

- All nodes are assigned a LAN ID.
- Packets which do not belong to the network specified by a node's LAN ID are discarded by the MAC layer and are not passed to the bridging layer.
- In any enabled state, all nodes store information derived from received HELLO.response packets.
- HELLO.response packets with a cost-to-root field value of 0 are from the root node.
- All nodes remain awake in any enabled unattached state. If SLEEPING nodes are unable to attach to the network after a maximum awake time has expired, an error is returned to a higher layer, and retries are handled by the higher layer.
- Root candidates must have a direct-link host port.
- All root candidates should be wired together.
- A node loses its address and goes into an intermediate hold-down state whenever a new root node is detected.

State Notation.

State names used in this document are hierarchical. Suffixes are added to state identifiers to further refine a state definition. An unqualified high-order prefix may be used to reference all possible substates.

The bridging entity in each node is in one of the following high-level **node states.**

R - Root node. The node owns the root node address.
RC - Root candidate node. The node does not have an address.
BB - Bridge node with a root priority of zero.
BR - Bridge node which has a non-zero root priority.
B - Any Bridge node.
T - Terminal node.
* - any node.

a - subscript used to qualify a node state to indicate that the node does not have a network address. For example, T_a is used to specify a terminal node that does not have a network address.

All node states are further qualified by one of three **attach states**:

D - The node is Disabled and unattached.
U - The node is enabled and Unattached.
A - The node is enabled and Attached to the network.

A node can be in an intermediate hold-down state:

I - The node is in an Intermediate hold-down state. Bridge nodes, which were attached, transmit HELLO.response packets with an infinite cost in the intermediate state.

As an example, RC.U, is used to denote the node and attach state of a root candidate which is not attached to an SST network.

The following substates are used to qualify an unattached node:

idle - No network activity has been detected.
wait - Wait for the first HELLO.response packet.
hello - A HELLO.response packet has been received.
rarp - An address request is in progress.
attach - An attach request is in progress.

Root Resolution Protocol.

Each SST network must have one or more root candidates. Each root candidate node enters a RC.U state when the bridging entity in the node is enabled. This state ends when 1) the root candidate determines that a higher priority root node already exists and enters the BR.U state, or 2) the root candidate assumes ownership of the root node status and enters the R.A state. A node in any BR state assumes the root node status if 1) the network becomes idle, or 2) a lower priority root node is detected.

A root candidate which does not detect any activity assumes the root node status. If activity is detected, the root candidate remains in the RC.U.wait state until a HELLO.response packet is received or until network activity ceases.

When an unattached non-candidate node receives its first HELLO.response packet it enters the *.U.hello state, and sets a hello-timer used to measure a HELLO learning period. The hello-timer expires after HELLO_RETRY HELLO periods.

In the R.A state the root node broadcasts a HELLO.response packet once per HELLO_PERIOD time period, according to a random distribution algorithm. The root HELLO.response packets contain a path cost of 0, the priority of the root node, a root ID sequence number, and a ROOT ID which is either the unique long identifier or the unique alias of the root device. **The pri rity, ROOT ID sequence, and ROOT ID fields are copied into the HELLO.response packets transmitted by all non-root bridges in the network.**

A ROOT ID sequence number is stored in non-volatile storage by all root candidates. The sequence number is copied into RAM by the root node when it determines that it is the root and the copy in non-volatile storage is incremented. The copy in RAM is included in all HELLO.response packets broadcast by all attached bridge nodes in the spanning tree.

Hello packet priority.

A "higher priority HELLO.response packet" is defined as any HELLO.response packet which contains a matching LAN ID and either 1) a higher concatenated USER PRIORITY+DEVICE PRIORITY field, or 2) an equal priority field and a higher priority ROOT ID. A ROOT ID can consist of a unique 48-bit device ID and/or a device alias. A "higher priority ROOT ID" is defined as 1) the ID with the higher 48-bit ID, or, 2) if neither candidate has a 48-bit ID, the ID with the alias with highest string value. Note that if the ROOT ID does not contain a unique 48-bit device ID, then the 48-bit device ID is assumed to be all 0's.

It may be possible for a root candidate to receive a HELLO.response packet with an equal priority if the ROOT ID field in the HELLO.response packet matches the candidate's device identifier. HELLO.response packets with a ROOT ID field that matches the identifier of the local device and a non-zero path cost are assumed to be associated with an out-of-date spanning tree and are discarded by the bridging layer. A matching ROOT ID and a zero path cost causes a fatal error.

Root resolution state transition table.

The state transition table below defines transitions in the root resolution process.

state	event	action	next state
RC.D	Bridging entity enabled.	Enable MAC on all network ports; set HELLO_TIMEOUT inactivity-timer.	RC.U.idle
RC.U.idle	Inactivity-timer expires.		R.A
	Non-HELLO packet received.	Set R_IDLE_TIME inactivity-timer.	RC.U.wait
	Higher priority HELLO packet received.	Set R_IDLE_TIME inactivity-timer; set HELLO_TIMEOUT hello-timer.	BRa.U.hello
	Lower priority HELLO packet received.		R.A
RC.U.wait	Inactivity-timer expires.		R.A
	Non-HELLO packet received.	Set R_IDLE_TIME inactivity-timer.	RC.U.wait
	Higher priority HELLO packet received.	Set R_IDLE_TIME inactivity-timer; set HELLO_TIMEOUT hello-timer.	BRa.U.hello
	Lower priority HELLO packet received.		R.A
BR.U	Lower priority HELLO packet received.	Set R_IDLE_TIME inactivity-timer.	BR.I then RC.U.wait
R.A	Higher priority HELLO packet received.	Transmit HELLO packets with an infinite path cost for MAX_HELLO_LOST+1 HELLO periods; set R_IDLE_TIME inactivity-timer; set HELLO_TIMEOUT hello-timer.	R.I then BRa.U.hello

Attaching to the Spanning Tree.

Nodes learn the best path to the root node by listening to HELLO.response packets from attached bridge nodes. After a learning time expires, a node without a network address sends a RARP.request packet to the attached bridge, in the in-range list, which provides the best path to the root. A node with a network address sends an ATTACH.request packet to the attached bridge in the in-range list which provides the best path. The "best path" is primarily a function of the path cost to the root.

Attach Selection Criteria.

The criteria for selecting a parent bridge node as the first hop in a best path is defined as follows: 1) first, only nodes which are in the in-range list, and not in the out-of-range list, are considered; 2) if no such node exists, then nodes which are in the in-range list but are classified as AGED, in the out-of-range list, are considered; 3) the node with the lowest cost to the root is chosen from the set of "parent candidates"; 4) if two or more nodes have the same path cost, then the node with the best signal strength is chosen (if a signal strength indicator is available); 5) if both the path cost and signal strength are equal, then the node with the highest attach priority is chosen; 6) if the path cost, signal strength and attach priority are equal, then the node with the lowest network address is chosen. Elements 5 and 6 of the selection criteria are intended to group terminals into logical coverage areas under the control of a single bridge node, rather than allowing terminals to randomly attach to any bridge node when physical coverage areas overlap.

All parent candidates must be in the in the in-range list. New entries in the out-of-range list are not considered as parent candidates. If no parent candidates exist an unattached node can wait and listen, or, optionally, can solicit short HELLO.response packets by transmitting a global HELLO.request packet.

Bridge Attach State Transitions.

The state transition table below defines the state transitions required for a disabled bridge node to obtain a network address. Note that a root candidate enters the B_n.U.hello state after learning of a higher priority root.

state	event	action	next state
BB.D	Bridging layer is enabled.	Enable MAC on all network ports; initialize in-range list to NULL; initialize out-of-range list to NULL.	B_n.U.wait
B_n.U.wait	HELLO packet received from the root.	Send RARP.request to the root; set B_TIMEOUT path-timer; set retry count to 1.	B_n.U.rarp
	HELLO packet received from a non-root bridge.	Set hello-timer.	B_n.U.hello
B_n.U.hello	Hello-timer expires or HELLO packet received from the root node.	Send RARP.request to the attached bridge with the best path; set B_TIMEOUT path-timer; set retry count to 1.	B_n.U.rarp
B_n.U.rarp	RARP.response packet received.	If address was received send ATTACH.request to the attached bridge with the best path; set B_TIMEOUT timer; set retry count to 1.	If address was received then goto B.U.attach else post error; goto BB.D or RC.D.
	path-timer expires.	Resend RARP.request. Increment retry count.	B_n.U.rarp
	Retry count > MAX_RARP_RETRY.	Initialize in-range list to NULL.	B_n.U.wait
	MAC layer retry error.	Add respective bridge to out-of-range list. Delete bridge from in-range list. If list is not empty, send a RARP.request to the attached bridge with the best path; set B_TIMEOUT timer; set retry count to 1.	If in-range list is empty then goto B_n.U.wait else goto B_n.U.rarp

The transition table below defines the state transitions required for an unattached bridge node, with a network address, to attach to the network. Note that an attached bridge node enters the B.U.wait state after detaching from the network. An unattached bridge node enters the B.U.attach state after receiving its network address.

state	event	action	next state
B.U.wait	HELLO packet received from the root.	Send ATTACH.request to the root; reset B_TIMEOUT timer; set retry count to 1.	B.U.attach
	HELLO packet received from a non-root bridge.	Reset hello-timer.	B.U.hello
B.U.hello	Hello-timer expires or HELLO packet received from the root node.	Send ATTACH.request to the attached bridge with the best path; reset B_TIMEOUT timer; set retry count to 1.	B.U.attach
B.U.attach	ATTACH.response packet received from bridge with best path in in-range list.		B.A
	ATTACH.response packet received from bridge other than one with best path.	(ignore)	B.U.attach
	B_TIMEOUT timer expires.	Resend ATTACH.request. Increment retry count.	B.U.attach
	Retry count > MAX_ATTACH_RETRY.	Initialize in-range list to NULL.	B.U.wait
	MAC layer retry error.	Delete bridge from in-range list. If list is not empty, send an ATTACH.request to the attached bridge with the best path; set B_TIMEOUT timer; set retry count to 1.	If in-range list is empty then goto B.U.wait else goto B.U.attach
	HELLO packet received with shorter path cost and not in out-of-range list.	Send ATTACH.request to the attached bridge with the best path; reset B_TIMEOUT timer; set retry count to 1.	B.U.attach

Note that the ATTACH.request packets, specified in the above transition tables, require an end-to-end response (i.e. an ATTACH.response packet).

State transition tables for an unattached bridge node which has a non-zero priority (BR.U) are identical to the above state tables for bridges with a priority of zero, with the following exceptions: 1) An unattached bridge with a non-zero priority assumes the root node status whenever no packets are received within an R_IDLE_TIME inactivity period in any state; 2) an unattached bridge with a non-zero priority enters the RC.U.wait state, after a BR.I hold-down period, whenever a lower-priority HELLO message is received. Note that an inactivity-timer is constantly running and must be reset whenever a packet is received.

In any attached or unattached state, a bridge loses its address, waits for a hold-down period, and then goes to a *.U.wait state, whenever a new root node is detected.

Terminal Attach State Transitions.

The state transition table below specifies the state transitions required for a disabled terminal node to attach to the network.

state	event	action	next state
T.D	Bridging layer is enabled.	Enable MAC on all network ports; initialize in-range list to NULL; initialize out-of-range list to NULL; set HELLO_TIMEOUT hello-timer.	T_a.U.wait
T_a.U.wait	HELLO packet received from the root.	Send RARP.request to the root; set B_TIMEOUT timer; set retry count to 1.	T_a.U.rarp
	HELLO packet received from a non-root bridge.	Reset HELLO_TIMEOUT hello-timer.	T_a.U.hello
	Hello-timer expires.	(sleep); reset HELLO_TIMEOUT hello-timer.	T_a.U.wait
T_a.U.hello	Hello-timer expires or HELLO packet received from the root node.	Send RARP.request to the attached bridge with the best path; set B_TIMEOUT timer; set retry count to 1.	T_a.U.rarp
T_a.U.rarp	RARP.response packet received.	If address was received send ATTACH.request to the attached bridge with the best path; set B_TIMEOUT timer; set retry count to 1.	If address was received then goto T.U.attach else post error; goto T.D
	B_TIMEOUT timer expires.	Resend RARP.request. Increment retry count.	T_a.U.rarp
	Retry count > MAX_RARP_RETRY.	Initialize in-range list to NULL.	Ta.U.wait
	MAC retry error.	Add respective bridge to out-of-range list. Delete bridge from in-range list. If list is not empty, send a RARP.request to the attached bridge with the best path; set B_TIMEOUT timer; set retry count to 1.	If in-range list is empty then goto T_a.U.wait else goto T_a.U.rarp

In any attached or unattached state, a terminal loses its address, waits for a hold-down period, and then goes to the T_a.U.wait state, whenever a new root node is detected.

The transition table below defines the state transitions required for an unattached terminal node, with a network address, to attach to the network. Note that an attached terminal node enters the T.U.wait state after detaching from the network. If the terminal has a message pending, then the terminal will solicit short HELLO.response packets with a global HELLO.request, and will wait HELLO_WAIT seconds for a HELLO packet.

state	event	action	next state
T.U.wait	HELLO packet received from the root.	Send ATTACH.request to the root; set B_TIMEOUT timer; set retry count to 1.	T.U.attach
	HELLO packet received from a non-root bridge.		T.U.hello
	Hello-timer expires.	Pause (and sleep); send global HELLO.request; increment retry count ; Reset S_HELLO_WAIT hello-timer.	T.U.wait
	Hello-timer expires; retry count > MAX_HELLO_WAIT .	Return uncompleted requests with error.	Ta.U.wait
T.U.hello	Hello-timer expires or HELLO packet received from the root node.	Send ATTACH.request to the attached bridge with the best path; set B_TIMEOUT timer; set retry count to 1.	T.U.attach
T.U.attach	ATTACH.response packet received from bridge with best path in in-range list.	Set current_parent to best path bridge.	T.A
	ATTACH.response packet received from bridge other than one with best path (i.e. in response to an old request).	(ignore)	T.U.attach
	B_TIMEOUT timer expires.	Resend ATTACH.request; increment retry count.	T.U.attach
	Retry count > MAX_ATTACH_RETRY.	Initialize in-range list to NULL; send global HELLO.request; set retry count to 1; Set S_HELLO_WAIT hello-timer.	T.U.wait
	MAC layer retry error.	Move bridge from in-range list to out-of-range list. If in-range list is empty, send a global HELLO.request; set S_HELLO_WAIT hello-timer; set retry count to 0; else, if in-range list is not empty, send an ATTACH.request to the attached bridge with the best path; set B_TIMEOUT path-timer; set retry count to 1.	If in-range list is empty then T.U.wait else T.U.attach

	Better path found.	Send ATTACH.request to the attached bridge with the best path: set B_TIMEOUT path-timer; set retry count to 1.	T.U.attach

In any attached or unattached state, a terminal loses its address, waits for a hold-down period, and then goes to the T_a.U.wait state, whenever a new root node is detected.

Reliable Flooding Mechanism.

Information can be flooded throughout the network by using a "reliable flooding mechanism". A node can flood a bridging layer request packet 1) by setting the MAC layer and bridging layer destination addresses to all 1's (i.e. a global address), 2) by setting the RSPRQ bit ON, and 3) by including a **forward list** in the packet. The forward list is identified by the bridging layer parameter type hexadecimal 08. It is used to specify which nodes must forward and acknowledge the request. Initially, the forward list consists of all bridge nodes which are either children or the parent of the node which generated the packet. Nodes in the forward list acknowledge the request packet with a unicast response packet of the same packet type. Note that only one flooded packet of a given type may be outstanding at a time. If a receiving node in the forward list is attached to one or more branches of the spanning tree, other than the branch on which the request packet arrived, then the node must rebroadcast the request packet. The forward list in the rebroadcast packet will consist of each bridge node which is the first hop on such other branches. Note that the forward list may be empty if the first (and only) hop in all other branches is a terminal node.

Spanning Tree Link Maintenance and Recovery.

A link in the spanning tree is lost whenever one of the following events occurs:

1) A child node is unable to deliver a message to its parent bridge node.
2) A child node finds a better path to the root node.
3) MAX_HELLO_LOST consecutive HELLO.response packets from a parent bridge node are lost by a child node.
4) A HELLO.response packet, from a parent bridge node or current root node, with an infinite path cost, is received by a child node.
5) A HELLO.response packet is received with a new ROOT ID or Root Sequence Number.
6) An attached node, which is not in a "DETACH HOLD_DOWN" state, receives a DETACH packet or HELLO.response packet with its address in the packet's detached node list.
7) A parent node is unable to deliver a message to a child node.
8) A routing table entry is aged and deleted.

- Response actions to events 1-6 are defined in the detach state transition tables below.
- Event 7 is discussed in the section which describes detach packet logic.
- Event 8 is discussed in the section on routing table maintenance.

Detach State Transitions.

Terminal Detach State Transitions.

The transition table below defines events which may cause a terminal to detach from the network.

state	event	action	next state
T.A	MAC_send error.	Remove old entries and the destination bridge from the in-range list; initialize the out-of-range list to the MAC destination bridge; If the in_range list is empty, send global HELLO.request; set S_HELLO_WAIT hello-timer; set retry count to 1; else, if in-range list is not empty, send an ATTACH.request to the attached bridge with the best path; set B_TIMEOUT path-timer; set retry count to 1.	If in-range list is empty then T.U.wait else T.U.attach
	Terminal address seen in a detached node list in a DETACH packet or HELLO packet and current time is past HOLD_DOWN.	Remove old entries from the in-range list; add the source of the DETACH or HELLO packet to the in-range list; initialize the out-of-range list to NULL; send an ATTACH.request to the attached bridge with the best path; set B_TIMEOUT path-timer; set retry count to 1; set HOLD_DOWN to current time plus HOLD_DOWN_TIME.	T.U.attach
	Better path found to the root and no data transactions are pending.	Issue a MAC_enquiry to the best path bridge.	T.A
	Positive response to a MAC_enquiry received from the best path bridge.	Send an ATTACH.request to the destination bridge.	T.U.attach
	Negative response to a MAC_enquiry.	Move the destination bridge from the in-range list to the out-of-range list.	T.A

	MAX_HELLO_LOST consecutive HELLO packets from the parent node are missed. (Note that sleeping terminals must count skipped HELLO times as misses.)	Send an ATTACH.request to the attached bridge with the best path: set B_TIMEOUT path-timer; set retry count to 1.	T.U.attach
	HELLO packet received from parent with an infinite path cost (same ROOT ID).	Remove parent and old entries from the in-range list; remove old entries from the out-of-range list; set HELLO_TIMEOUT HELLO timer.	T.U.wait
	HELLO packet received with new ROOT ID or Root Sequence Number.	Update current-root. Initialize node identifier to all 1's. Initialize in-range and out-of-range lists to NULL.	T_R.U.wait

A sleeping terminal should remain awake, whenever a threshold number (i.e. 1 or 2) consecutive HELLO packets have been missed from its parent node, until a HELLO packet is received from its parent or until the terminal detaches and reattaches to a different parent node.

A terminal which is waiting to receive a response time critical message can periodically transmit an empty DATA.request packet to its parent bridge node, to determine if the parent is still in range, if the message is not received in the expected amount of time.

A terminal which is in a DETACH HOLD_DOWN state must send an ATTACH.request packet to the root node after the HOLD_DOWN state ends, to ensure that it is fully attached to the network.

Bridge Detach State Transitions.

The transition table below defines events which may cause a bridge node to detach from the network. The "detach" action, specified in the table, consists of broadcasting infinite cost HELLO packets for MAX_HELLO_LOST+1 HELLO periods, in an intermediate detach (B.I.detach) state before going to an unattached state.

state	event	action	next state
B.A	MAC_send error.	Detach. Remove the MAC destination and old entries from the in-range list; initialize the out-of-range list to the MAC destination bridge; set HELLO_TIMEOUT hello-timer.	B.U.wait
	Bridge address seen in a detached node list in a DETACH packet or HELLO packet.	Detach. Remove old entries from the in-range list; add the source of the DETACH or HELLO packet to the in-range list; initialize the out-of-range list to NULL; set HOLD_DOWN_TIME hello-timer.	B.U.wait
	Better path found to the root.	Issue a MAC_enquiry to the best path bridge.	B.A
	Positive response to a MAC_enquiry received from the best path bridge.	Send an ATTACH.request with a decedent list to the destination bridge.	B.U.attach
	Negative response to a MAC_enquiry.	Move the destination bridge from the in-range list to the out-of-range list.	B.A
	MAX_HELLO_LOST consecutive HELLO packets from the parent node are missed.	Detach. Remove old entries from the in-range list; initialize the out-of-range list to the parent node; set MAX_HELLO_PERIOD hello-timer.	B.U.wait
	HELLO packet received from parent with an infinite path cost.	Detach. Remove parent and old entries from the in-range list; remove old entries from the out-of-range list; set HELLO_TIMEOUT HELLO timer.	B.U.wait
	HELLO packet received with new ROOT ID or Root Sequence Number and either bridge can not be a root candidate or packet has a higher priority.	Detach. Update current-root. Initialize node identifier to all 1's. Initialize in-range and out-of-range lists to NULL.	B_a.U.wait
BR.A	Lower priority HELLO packet received.	Detach. Initialize in-range and out-of-range lists to NULL.	R.A

Note that a bridge node will never be in an attached HOLD_DOWN state.

Detach packet l gic.

A parent node generates a DETACH.request packet whenever it is unable to deliver a message to a child node. The two possible cases are 1) the child is a bridge, or 2) the child is a terminal.

Case 1 - The lost node is a bridge.

When a parent bridge node is unable to deliver a message to a child bridge node, it must send a DETACH.request packet, to the root node, which contains a detached node list that describes the lost subtree. The list contains all nodes in the routing table of the parent for which the lost bridge was the first upstream hop. All downstream bridge nodes in the path of the DETACH.request packet must adjust their routing tables by deleting entries which match those in the detached list. Detached node information is copied into a bridge's "detached node list".

Case 2 - The lost node is a terminal.

Since terminals can be mobile they can be lost often and must be notified quickly. When a parent node is unable to deliver a message to a terminal, it must generate a DETACH.request packet, with the terminal specified in the associated detached node list, and flood the packet throughout the network using the reliable flooding mechanism described above. Any bridge node, which receives the DETACH.request, adds the detached terminal to its internal detached node list.

A bridge node in the forward list does not forward an entry in the detached list of a DETACH.request if the DETACH.request came from an upstream node, and the upstream node is not the first hop in the routing table entry associated with the entry in the detached list. A DETACH.request is discarded if the detached list becomes empty.

Upstream bridges, which only have terminals nodes (i.e. do not have bridge nodes) as children, must convert a received DETACH.request to an unscheduled HELLO.response packet (without a forward list) and broadcast it immediately. Therefore, lost node information is flooded throughout the coverage area of the spanning tree. An awake unattached terminal should quickly discover that it has been detached and can then reattach.

Each entry in a bridge node's detached list is advertised in HELLO.response packets for MAX_HELLO_ LOST+2 HELLO times or until the bridge determines the terminal has reattached. The same entry can not be in a bridge's detached list for a hold-down time after it was deleted from the list.

Since a node assumes that it is detached if it does not receive a HELLO. response packet from its parent with MAX_HELLO_LOST HELLO time periods and detached node information is copied in HELLO.response packet for MAX_HELLO_LOST+2 HELLO time periods, a lost node is guaranteed that it will always discover when it becomes unattached within MAX_HELLO_LOST*HELLO_PERIOD seconds, in the worst case.

Unscheduled HELLO.response messages are generated in each bridge node if a new detached bridge is discovered and the time until the broadcast of the next scheduled HELLO.response packet is greater than HELLO_TRIGGER seconds.

Network Routing.

A routing table is maintained in each bridge node. The routing table has an entry for each known upstream node. (Downstream routing is defined by the structure of the spanning tree.)

An example routing table is shown below:

destination	port	first hop	age	child flag	delivery service	awake begin time	awake end time
hex 080A	1	hex 020A	0	false	null	null	null
hex 020A	1	hex 020A	0	true	null	null	null
hex 080B	1	hex 080B	1	true	1	15320	15330

In the example route table above, the bridge has three descendants - 080A, 020A, and 080B. 080A and 080B are terminal nodes and 020A is a bridge node. (Note that the node type can be determined by the address.) If the first hop field is the same as the destination field, then the (optional) child flag field is set to true. The delivery service, awake begin time, and awake end time fields only apply to child terminal nodes with a non-zero delivery service field.

To forward an upstream bridging layer packet, a bridge node looks up the entry associated with the destination address in the bridge header. If the entry does not exist, the packet is discarded. If the entry does exist, the MAC layer destination address is set to the first hop address in the route table. The MAC source address is set to the address of the bridge node. (The bridge header addresses remain unchanged.) The packet is passed to the MAC entity on the port specified in the table.

Route table entries are created or updated whenever a downstream unicast DATA, ATTACH, or ARP packet is received. If an entry does not exist for the bridging layer source address, an entry is created with the destination field set to the bridging layer source address. The fields in the (old or new) entry for the destination are modified as follows: 1) the first hop field set to the MAC layer source address, 2) the port field is set to identify the MAC entity which delivered the packet, 3) the age field is set to 0, and 4) if the destination and first hop fields are identical, the child flag field is set to true.

The age field for each entry is incremented at regular intervals. An entry's age field is reset to 0 whenever a packet is received from the destination associated with the entry. If no packets are received from the destination of an entry for ROUTE_TIMEOUT seconds, the entry is deleted from the route table.

Nodes can maintain their path in the network by sending an ATTACH.request packet to the root node once every ATTACH_TIMEOUT seconds, where ATTACH_TIMEOUT must be shorter than ROUTE_TIMEOUT. The ATTACH.request packet can be piggybacked on a higher-layer data packet, if the node is active.

If a DETACH.request packet is received from an upstream bridge node, then each entry in the route table, with a destination field which matches an entry in the packet's detached list, is deleted.

All nodes must maintain a **current_parent** structure. The structure has a network address field, a port field, and an age field. The age field is incremented once per HELLO period. If the count stored in the age field reaches MAX_HELLO_LOST, the current_parent structure is

re-initialized to null values, and the node becomes detached. The age field is reset to 0 whenever a scheduled or unscheduled HELLO.response packet is received from the node's parent. A downstream packets is forwarded by setting the MAC destination address to the current_parent.address and then passing the packet on the current_parent.port.

All nodes must maintain a **current_root** structure. The structure has an age field, a 48-bit long root ID field, a root alias field, and a root sequence field. The structure is initialized to all 0's (or null). The structure is checked, and possibly updated, each time a HELLO.response packet is received. If a node receives a HELLO.response packet which contains a long ID, alias, or root sequence number which is different than the associated value contained in the structure, then the node becomes unattached and loses its network address. The age field in the current_root structure is incremented periodically. If NETWORK_TIMEOUT minutes expire before a node receives a HELLO.response packet from any bridge node, then the current_parent structure is re-initialized and the node loses its address. The age field is reset to 0 whenever a HELLO.response packet is received from any bridge node and the root node has not changed.

Address Resolution and Maintenance.

Reverse Address Resolution Protocol (RARP) protocol.

An address server in the root node maintains network addressing information in an address table, distributes network addresses to requesting nodes, and resolves network addressing problems. Each entry in the address table contains a device type field, a network address field, a long ID field, an alias field, an in-use field, and an age field. Entries in the table are aged so that they can be reused after MAX_ADDRESS_LIFE minutes. Note that entries may remain in the table indefinitely. The age field in an entry is reset to 0 whenever a RARP.request or ATTACH.request packet is received from the node associated with the entry.

A separate sequential set of unique node identifiers is maintained for each device type. Each set begins with an identifier of 1 and ends with the maximum range for the device type. Lower node identifiers are distributed before higher identifiers.

A RARP.request packet can be used to: 1) acquire a network address from the address server, 2) change an existing 48-bit long ID in the address table, or 3) change an existing alias in the address table.

A node which does not yet have a unique 16-bit network address must request a 11-bit node ID from the address server. The node uses a 16 bit multicast address until a unique node ID is assigned. The low order 15 bits of the temporary address consist of the node's device type concatenated with an 11-bit node ID of all 1's. A RARP.request packet, containing the requesting node's unique 48-bit long ID and/or an alias, is sent to the address server by the requesting node. When a node requests a new address, the server first checks its address table to determine if the node already has a valid address. If the node doesn't already have an address, the server allocates the first available node identifier for the device type, to the node, if one is available. In either case, if an address is available, the server will set the network address field in the RARP packet to the allocated address and will set the return code bits to 0. If an address is not available, or an entry already exists in the address table with ambiguous identifiers, the address server will set the network address field to all 1's and will indicate the error in the return code field.

The long identifier and/or alias in a RARP.request packet matches an entry in the address table 1) if matching 48-bit IDs and aliases are present in both, 2) if both contain a matching alias and neither has a 48-bit ID , or 3) if both contain a matching 48-bit ID and neither has an alias. If either the 48-bit ID or alias in the request packet is also in the address table, and the above criteria are not met, an error is returned.

Occasionally, a node may want to change the association between a 48-bit ID and an alias. If the node simply requests a new address, and its old address has not expired, an error will be returned. The requesting node can override the error and force a change by setting the New Alias or New Long ID bits in the RARP.request operation field. A change operation causes a new address table entry to be created if neither the long ID or alias can be found in the address table.

A requesting node can specify that it does not have a 48-bit ID by not including it as a parameter or by including an ID of all 0's. An alias is required if there is no 48-bit ID. A requesting node can specify that it does not have an alias by not including it as a parameter, or by including an alias with a length of 0.

The address server will return a RARP.request packet to the requesting node as a RARP.response packet. If the node, which generated the RARP.request packet, does not receive a RARP.response packet within BRIDGE_TIMEOUT seconds, it must resend the RARP.request.

RARP hexadecimal 4-bit error codes.

0 = good.
1 = a node ID is not available.
2 = Duplicate alias. The alias specified in the RARP.request packet is already in the address table with a different long ID.
3 = Duplicate long ID. The long ID specified in the RARP.request packet is already in the address table with a different alias.
4 = Invalid device type.
F = Extended error code.

RARP routing.

Each bridge node maintains a RARP routing table which contains entries for upstream nodes which have recently sent a RARP.request packet to the root node.

An example RARP route table is shown below:

long ID	alias	port	first hop	network address	return code	age
hex 1003508A990C	null	1	hex 020A	hex FFFF	invalid	0
hex 1003508A920B	term2	1	hex FFFF	hex 080C	0	3

Whenever a RARP.request packet is received, an entry is created (or updated) in the RARP route table and the long ID and/or alias fields in the entry are set to the values specified in the request packet. **The node which initiated the request is defined by the l ng ID and/or alias.** The long identifier and/or alias in a RARP.request packet matches an entry in the RARP route table 1) if matching 48-bit IDs and aliases are present in both, 2) if both contain a

matching alias and neither has a 48-bit ID , or 3) if both contain a matching 48-bit ID and neither has an alias. The return code is initialized to invalid to indicate that an associated RARP.response packet, destined for the node which originated the RARP.request, has not been received. The port field points to the port on which the RARP.request was received. The network address is set to the bridging layer source address of the RARP.request packet. (Normally, the node ID in the bridging layer source address will be all 1's, which is the default global node ID used before a unique network node ID is obtained. If a node is attempting to change its long ID or alias, then the network address may be unique.) The first hop field will be set to the MAC source address. The age field will be set to 0.

Normally, a bridge node will forward RARP.request packets to the root node on the port specified in the current_parent structure. However, if a bridge node receives a RARP.request packet, and 1) an entry for the node which initiated the request is already in the RARP route table, 2) the entry has a return code field which is valid, and 3) the network address field is not all 1's, then the bridge can simply return a RARP.response packet to the source node. The return code and age fields in the route table entry are not modified. The network address and return code fields in the RARP.response packets are set to the values contained in the RARP route table.

When a bridge node receives a RARP.response packet from the root node, it will update the return code and network address fields in the RARP route table entry for the node which initiated the request. RARP.response packets are forwarded on the port specified in the route table entry. The MAC destination address is set to the first hop address.

RARP route table entries are aged (quickly) so that older entries are discarded in RARP_TIMEOUT seconds.

Address Resolution Protocol (ARP) protocol.

A node can request the 16-bit network address of another node by sending an ARP.request packet to an address server in the root node. The ARP.request packet must contain either the 48-bit identifier or the alias of the target node, but not both. The address server returns the 16-bit network address of the target node in an ARP.response packet, if the target node exists in the server's address table. An address of all 1's and an error is returned if the target node is not in the address table.

ARP 4-bit hexadecimal error codes.

0 = good.
1 = long ID not found.
2 = alias not found.
F = extended error code.

Address Maintenance.

A node will lose its address if:

1) The root node changes (i.e. either a different ROOT ID or ROOT ID Sequence Number is detected in a HELLO.response packet).

2) It has not received an ATTACH.response packet from the root node, within an ADDRESS_TIMEOUT time period.

3) No network activity is detected within a NETWORK_TIMEOUT time period.

The root node can change because 1) the root node is lost or 2) a new root is chosen with the root selection protocol. All bridge nodes which detect the change must broadcast HELLO.response packets at scheduled times with the new ROOT ID and an infinite path cost for MAX_HELLO_LOST+2 HELLO times. If the root node is lost the new ROOT ID will be either a 48-bit long identifier of all 0's, if the old ROOT ID was a long identifier, or a null alias if the old ROOT ID was an alias. If the a new root is chosen the new ROOT ID will be either the 48-bit long identifier or alias of the new root node. Note that a new occurrence of the same root node can be always be detected because the ROOT ID sequence number will change.

A node can maintain its address by sending an ATTACH.request packet to the root node at least once per ADDRESS_TIMEOUT time period. Note that a node must send an ATTACH.request to the root at least once per ROUTE_TIMEOUT time period, to maintain its path to the root in the spanning tree; therefore no special logic is required for address maintenance. If the node is active it can simply piggyback the ATTACH.request on a higher-layer downstream data packet. The root node will return an ATTACH.response packet, and the node can reset its ADDRESS_TIMEOUT timer when the response packet is received.

HELLO Timing.

HELLO.response packets can be unscheduled or scheduled.

Unscheduled HELLO timing.

Unscheduled HELLO.response packets are broadcast in response to HELLO.request packets.

In addition, an unscheduled HELLO.response packet is generated in each bridge node if a new detached bridge is discovered and the time until the broadcast of the next scheduled HELLO.response packet is greater than HELLO_TRIGGER seconds.

Scheduled HELLO timing.

Each attached bridge node broadcasts one scheduled HELLO.response packet per HELLO_PERIOD seconds. Each HELLO_PERIOD time period is divided into AVG_HELLO_SLOTS slots, where each slot is HELLO_SLOT_SIZE seconds. Initially, a bridge node chooses a slot and broadcasts a HELLO.response packet. The HELLO.response packet contains a "seed" field which is used in a well-known "hello randomization algorithm" to determine the next hello slot and the next seed for the bridge node's next HELLO.response packet. The algorithm guarantees that a bridge node will randomly broadcast its next HELLO.response packet within a MIN_HELLO_PERIOD second to MAX_HELLO_PERIOD second time window after its last HELLO transmission.

Since significant cumulative delays in hello timing are prohibited, nodes can execute the hello randomization algorithm i times to determine the time (and seed) of the i-th successive scheduled HELLO.response packet from a bridge node. For example, a SLEEPING terminal node can initially synchronize on a HELLO.response packet from its parent. The terminal can calculate the time of a future HELLO.response packet from its parent and can power-down with an active timer interrupt set to wake it just before the broadcast of the HELLO.response packet is expected.

Contention delays incurred during the transmission of a HELLO.response packet are recorded in a "displacement" field in the packet. The displacement field specifies the transmission offset time, in hundredths of seconds, from the calculated transmission time. Note that errors, caused by rounding to the nearest displacement, can cause some drift in calculated transmission times. Therefore, nodes should resynchronize every time a HELLO.response packet is broadcast. If the displacement field, in a HELLO.response packet, is set to all 1's, then the transmission time of the packet does not coincide with a calculated hello time, and the packet should not be used for hello synchronization. The displacement is set to all 1's in unscheduled HELLO.response packets and is set to all 1's in scheduled HELLO.response packets whenever the packet can not be sent within MAX_HELLO_OFFSET milliseconds.

If an expected HELLO.response packet is not received at the calculated time, then the receiving node should wait for the maximum displacement time before assuming the HELLO packet was missed. The maximum displacement time can be calculated, in milliseconds, by multiplying the number of hello periods, since the last HELLO packet was received, by MAX_HELLO_OFFSET.

Note that HELLO slot boundaries for different bridge nodes are not necessarily aligned (i.e. as in slotted ALOHA). In fact, to increase randomization, bridge nodes should avoid slot alignment. A "HELLO slot" is simply the unit of time used in the randomization algorithm.

Default HELLO_PERIOD and HELLO_SLOT_SIZE values are set at compile time and are well-known by all nodes. Modified HELLO_PERIOD and HELLO_SLOT_SIZE values can be set by the root node and advertised throughout the network in HELLO.response packets.

The algorithm used for HELLO time calculation is defined by the "C" routines below. The routines assume that *last_hello_time* and *last_hello_seed* variables are maintained for each bridge node of interest. A bridge node should set the *last_hello_time* variable, associated with its HELLO.response packet broadcasts, to the last HELLO transmission time, adjusted to account for any displacement. For example, if a bridge node's HELLO transmission time is delayed approximately 20 milliseconds, then it should set the displacement field in the packet to 2 and sets its *last_hello_time* variable to the transmission time minus 20 milliseconds.

Two routines are shown below:

set_last_hello - is used to extract *last_hello_time* and *last_hello_seed* from a received HELLO. response packet. The last_hello_time variable is adjusted to account for the displacement field. The routine assumes that packets are accurately time-stamped (i.e. by the MAC layer) when they are received.

calc_next_hello - is used to calculate the time and seed of the next HELLO.response packet for the bridge specified by the address passed to the routine.

The routines rely on the address table functions - **Insert** and **Lookup**. **Insert** enters an address into a table, if it does not exist and returns the index. **Lookup** returns the index of an address, if it exists.

```
int set_last_hello_time(PACKET * hello_packet)
{
  offset=hello_packet->hello_seed & B_MASK_DISP;
  if (offset==B_MASK_DISP)
    return -1;  /* next hello time cannot be calculated, if the displacement is all 1's */

  /* else, determine the seed and hello time */
  i=Insert(hello_packet->mac_s_addr);
  last_hello_time[i]=hello_packet->receive_time  - ((TIME)(offset));
  last_hello_seed[i]=hello_packet->hello_seed >> 2;
  return 0;
}

int calc_next_hello(int address, TIME* next_time, int* next_seed)
{
  int next_slot;

  i=Lookup(address);
  if (i < 0) return -1;  /* next hello time cannot be calculated */

  *next_time=last_hello_time[i];
  *next_seed=last_hello_seed[i];
  while (*next_time <= current_time())
  {
    next_slot=((*next_seed+address) % HELLO_MOD_VAL) + HELLO_MOD_VAL;
    *next_time += next_slot * HELLO_SLOT_SIZE;
    *next_seed=((seed + 3) ^ 0x2A) & 0x3F;
  }
  return 0;
}
```

Sleeping terminal support.

The bridging layer provides a service which will store pending messages for SLEEPING terminals. Terminals request the service by setting the optional "delivery service" parameter (type hexadecimal 0E), in the optional parameter area in any bridging layer packet. If a child node does not set the parameter or sets it to 0 (the default value), then the parent bridge node will attempt to deliver packets to the child node immediately. The child node is assumed to be lost whenever a packet can not be delivered. If the parameter is set to 1, the source child node is assumed to be a SLEEPING terminal and the parent node will not attempt to deliver packets to the child node until the child notifies the parent that it is awake. If the parameter is set to 2, the parent bridge will attempt to deliver packets immediately, with a reduced retry count; if a packet can not be delivered, the parent will store the packet until the child node notifies the parent that it is awake (or a save time expires).

A child node can notify its parent node that it will be awake by setting an optional "awake time" parameter (type hexadecimal 0C) in any bridging layer packet. The awake time is specified in AWAKE_TIME_UNIT time units. A value of all 1's indicates forever. By default, the awake time period is assumed to follow the end of transmission of the packet containing the awake time parameter. Optionally, an offset can be specified with the optional "awake time offset" parameter (type hexadecimal 0D). The awake time offset parameter specifies an offset time at which time the terminal will wake up and stay awake for the time specified in the associated awake parameter. The offset is specified in AWAKE_TIME_UNIT time units and is added to the end-of-transmission time. This option allows a terminal to initiate a transaction, sleep for a statistically calculated time period, and then wake up to accept a response.

If a bridge node determines that a SLEEPING terminal is sleeping then it will store packets destined for the terminal for a number of HELLO periods, or until delivery to the child node is attempted. The number of HELLO periods that a packet will be saved is set to DEF_SAVE_CNT, by default, and can be set from 0 to MAX_SAVE_CNT by a child node with the optional "store message count" parameter (type hexadecimal 0F). Whenever a bridge node is storing a message for a terminal, the terminal's address will be included in the "pending message list" in successive HELLO.response packets broadcast by the bridge node. If the message can not be delivered within "store message count" HELLO periods then the bridge node will assume that the child terminal node is lost.

As an example, a SLEEPING terminal might notify its parent that it will be awake, in a time window when a reply message (i.e. from a host computer) is expected, by setting the awake time and awake time offset parameters in the packet which solicited the reply. If the reply message is not received when expected, the terminal can alternate between sleep periods and short awake time periods. An empty DATA.request packet containing a "wake time" parameter can be used to notify the parent of successive wake time periods. (Note that a parent node will generate an ATTACH.request packet with the ATTI bit set ON whenever a DATA.request packet is received from a terminal and a routing table entry does not exist for the terminal. Therefore, the empty DATA.request also ensures that the node is still attached to the network.) Whenever a SLEEPING terminal expects a reply message the terminal should wake up to receive all HELLO.response packets from its parent. If an expected message is not received during any of the awake periods, the terminal can simply continue to listen to scheduled HELLO.response packets. If a response message is expected and a threshold number (i.e. 1 or 2) of consecutive HELLO.response messages are missed, then the terminal should transmit a (empty) DATA.request packet to its parent to determine if it is still in range. Note that if the parent bridge fails to deliver a message to a child terminal then

either 1) the child will see its address in a "detached node list", or 2) will reattach to the network after a threshold number of HELLO.response messages are missed.

LLC implementation notes:

An LLC layer RR (receive-ready) packet, with the final bit set ON, should be transmitted by the terminal whenever the terminal attaches to the network. This ensures that higher layer data which may have been lost will be retransmitted immediately by the LLC entity at the remote end of the LLC connection.

Receive timeouts can be handled by the LLC entity in a terminal, if the bridging layer:

1) allows the LLC layer to pass an awake time window along with a data message.
2) provides a "request_message" service to the LLC layer. The request_message service causes the bridging layer to send an "awake_time" packet to the parent bridge node.
3) notifies the LLC layer whenever the terminal (re)attaches to the network.
4) provides a "net_monitor" service to the LLC layer. The net_monitor service causes the network layer to wake up for all HELLO packets from the parent bridge node, and causes the network layer to query the parent node (with an empty DATA.request packet) when a HELLO packet is missed.

An example algorithm for sending a message, for which a reply message is expected, using type 1 delivery service, is shown below:

```
awake_time_window->begin=minimum_data_delay;
awake_time_window->end=calculated_data_delay;
set_alarm(SHORT_TIMEOUT,
          awake_time_window->begin+awake_time_window->end+pause(0));
set_alarm(LLC_TIMEOUT, MAX_LLC_TIMEOUT);
net_send(destination, LLC_data, awake_time_window);*
net_monitor(TRUE);

while (waiting_for_response)
{
  event=wait(event_Q);
  if (event==SHORT_TIMEOUT && request_count++ < AWAKE_MAX)
  {
    request_message(awake_time);
    set_alarm(SHORT_TIMEOUT, pause(request_count));
  }
  else if (event==REATTACH)
  {
    awake_time_window->begin=0;
    awake_time_window->end=RR_TIMEOUT;
    request_count=AWAKE_MAX;
    reset_alarm(LLC_TIMEOUT, RR_TIMEOUT);
    net_send(destination, LLC_RR, awake_time_window);
  }
  else if (event==LLC_TIMEOUT && retry_count++ < LLC_MAX)
  {
    awake_time_window->begin=0;
    awake_time_window->end=RR_TIMEOUT;
    request_count=AWAKE_MAX;
    reset_alarm(LLC_TIMEOUT, RR_TIMEOUT);
    net_send(destination, LLC_RR, awake_time_window);
  }
  else
    waiting_for_response=FALSE;
}
net_monitor(FALSE);
set_alarm_off(SHORT_TIMEOUT);
set_alarm_off(LLC_TIMEOUT);
return event;
```

The while loop will terminate if 1) a reply message is received, 2) the maximum number of LLC timeouts is exceeded, or 3) a RR/RNR frame is received from the remote LLC entity. In the last case, the LLC state machine will either determine that the original LLC data frame must be retransmitted or that the LLC entity should continue to wait for a reply message.

*Note that the FINAL bit should be set OFF in the "LLC_data" frame so that an unnecessary RR frame will not be generated by the remote LLC entity.